J. Symbolic Computation (1997) **24**, 113–122

PSPCLink: A Cooperation Between General Symbolic and Poisson Series Processors

ALBERTO ABAD† AND FELIX SAN-JUAN‡

†*Grupo de Mecánica Espacial, Universidad de Zaragoza, 50009 Zaragoza. Spain*
‡*Grupo de Mecánica Espacial, Universidad de La Rioja, 26004 Logroño. Spain*

Poisson series appear frequently in problems of non-linear dynamics and celestial mechanics. The size of such mathematical objects makes their manipulation by means of general symbolic processors (GSP) inefficient. Special processors named Poisson series processors (PSP) have been created to handle them in a more efficient way. We propose here a way to combine the flexibility and easy use of GSPs with the power of PSPs. Using the communication tool *MathLink* we produced PSPCLink to connect *Mathematica* with PSPC, a PSP of our own creation. PSPCLink is a set of C-files and one *Mathematica* package. Compiling the C-files we created an executable program to be called from *Mathematica*. With PSPCLink we use the PSPC functions in a way completely integrated into *Mathematica* taking advantage of the good properties of both *Mathematica* and PSPC.

1. Introduction

In Celestial Mechanics, the enormous size of some tasks, such us lunar and planetary theories (Deprit *et al.*, 1970; Brumberg and Chapront, 1973), makes it imperative to write special algebraic manipulators oriented towards these particular problems. The formulation of the latter is based on multivariate Fourier Series whose coefficients are multivariate Laurent Series.

$$\sum_{\substack{i_0,\ldots,i_{n-1}\\ j_0,\ldots,j_{m-1}}} C^{j_0,\ldots,j_{m-1}}_{i_0,\ldots,i_{n-1}} x_0^{i_0}\ldots x_{n-1}^{i_{n-1}} \begin{pmatrix}\sin\\ \cos\end{pmatrix}(j_0y_0+\cdots+j_{m-1}y_{m-1}). \tag{1.1}$$

This mathematical object is commonly named a Poisson series (Deprit, 1990). The set $\mathcal{P}$ of a Poisson series is a commutative algebra with a unit element. This good algebraic structure of $\mathcal{P}$ combined with efficient algorithms makes it possible to design efficient Processors for dealing with them. Named Poisson series processors (PSP), this tool has been frequently used not only in celestial mechanics, but in other problems of non-linear dynamics (see Ricklefs *et al.*, 1983; Broucke, 1989; Henrard, 1989, for a historical review).

† E-mail: abad@posta.unizar.es

0747–7171/97/010113 + 10 $25.00/0 sy970116

Most of the classical PSPs have been written in FORTRAN. It is a high-level language but it does not allow data abstraction, i.e. programming using user-defined types (Stroustrup, 1987). The C language supports neither data abstraction nor object-oriented programming, but it is nearer than FORTRAN to these concepts of programming, and this is one of the reasons why new PSP written in C are now appearing. More detailed reasons are given in the next section together with the characteristics of a PSP of our own creation called PSPC.

General symbolic processors (GSP), like Macsyma, Maple, Axiom or *Mathematica* are becoming more and more useful in all branches of scientific work since they provide a powerful tool to analyse and solve a wide range of problems. For example, as one illustration Palacián (1993) produced the *Mathematica* package MALISIAS to generate analytical theories in the artificial satellite problem using the Poisson series.

The objects in *Mathematica* are represented by means of tree structures. This structure, combined with a system of evaluation based on the matching of patterns, makes *Mathematica* a flexible tool for studying a lot of different computational objects, from mathematical functions like `Sin[x]` or `ArcTan[x,y]`, to completely different ones like graphics `Plot[Sin[x],List[x,0,Pi]]`, etc.

With its structure *Mathematica* represents a Poisson series as a head `Plus` whose arguments are the terms of the series, but it does not distinguish between a Poisson series and other objects. As a consequence, it needs to store all the elements of the expression to handle them (Wolfram, 1988), while PSPs take advantage of the properties of *only the object* they handle to pack the information. In the usual problems of celestial mechanics there appear series with a very high number of terms; in this case, the memory becomes critical and PSPs are more efficient. The MALISIAS package showed the difficulty of obtaining high orders of the theories with *Mathematica*, while these orders can be obtained without difficulty using PSPC (Abad and San-Juan, 1993; San-Juan, 1996). However, sometimes the use of a Poisson series is not sufficient for developing a theory, and we need to add other special functions like natural logarithms, dilogarithmic functions (Osácar and Palacián, 1994), and other objects without a well-defined algebraic structure. In these cases PSPs cannot be used and we need a GSP.

Modern tools of communication between programs can help us to have a tool with the combined advantages of both PSP and GSP systems. Specifically *MathLink* (Wolfram, 1992) is a communication protocol for *Mathematica* that is a way of sending data and commands back and forth between *Mathematica* and other programs, in particular C programs. With *MathLink* we produced PSPCLink. *Mathematica* uses PSPCLink to call the routines of PSPC to perform all operations with a Poisson series. PSPCLink adds to *Mathematica* the efficiency of PSPC. *Mathematica* for its part adds to PSPC the flexibility and the possibility of using a great variety of *Mathematica* objects and its friendly front end.

The structure of a Poisson series suggests the use of object-oriented programming (OOP) to design PSPs, but at this moment we do not have the possibility of connecting *Mathematica* with an OO language like C++. *Mathematica* is not an OO language, however, we used the features of *Mathematica* to add OOP characteristics, like polymorphism, to PSPCLink.

2. PSPC

A few years ago, we decided to write our own program for manipulating a Poisson series. Up to that time, we had used the PSP of Dasenbrock (1982), and we grew out of it. We needed to improve its performance and we wanted to add new functions to it.

Rather than restructuring PSP, we decided to start from scratch. We had in mind principally to provide our colleagues and students at the Grupo de Mecánica Espacial with a common toolbox for them to build programs adapted to their own applications.

We adopted C as the programming language. Dasenbrock had written his PSP in FORTRAN, but times have changed since then. We concur with the general view that, nowadays, in the area of symbol manipulation, C is more suitable than FORTRAN and more portable than LISP. We found many advantages in C.

(i) C provides a preprocessor, a capability we exercise to parametrize PSPC before compilation by means of macrovariables. PSPC is more than a processor of Poisson series; at a higher level of abstraction it is an instrument for creating PSPs. In that perspective, the number of trigonometric arguments, the number of polynomial variables, the chopping threshold for real coefficients, the size of a block of memory for allocation, the output stream, all these parameters and a few other specifications are made into macrovariables. They must be set before compiling the code; every time a macrovariable is changed, the code must be recompiled. With macrovariables we enable the users to proportion the toolbox to their needs. Those, for instance, who deal only in pure polynomials will appreciate the savings they make in run time and memory from not having to handle them as a Poisson series in a fixed number of angles never to be zero.

(ii) Thanks to the mechanism of dynamic allocation, no slice of memory is reserved in advance for series storage. Requirements in memory grow and contract with the series. If the equipment allows for it, the program can take advantage without modification of the virtual address mechanism in the software and hardware of the host equipment.

(iii) Internal structuring is achieved by means of pointers to words; chaining by pointers permits scanning by hopping over terms in a series.

(iv) C lets the user define new types of data structures. We take advantage of this to introduce series, nodes for trigonometric arguments, polynomial exponents, numerical coefficients, etc., each of these entities associated with pointers in different directions.

A multivariate Fourier series of the form of equation (1.1) is called a Poisson series. In practice, in order to define an equivalent computational object, we need to consider only series with a finite number of terms. Each term of one of these series consists of a rational or real coefficient, n polynomial variables with integer exponents and a trigonometric function of a linear combination of angular variables. These elements are stored in three different kinds of nodes in a bidimensional structure (see Figure 1).

The computational object, called *series* by an abuse of language, contains information about the number of the three kinds of nodes and pointers addressed by the first and the last polynomial and trigonometric nodes. In particular, the zero element of the algebra $\mathcal{P}$ is represented by a *series* with one coefficient equal to 0, one polynomial node whose exponents are all 0 and a cosine of zero as a trigonometric node.

The polynomial part of the series is a sequence of n integer variables forming the

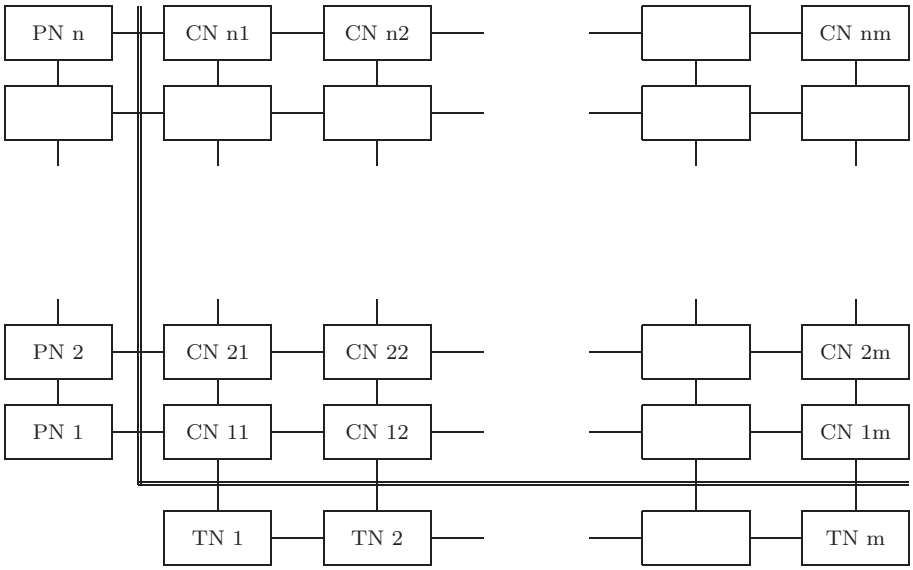

Figure 1. Bidimensional structure of PSPC. TN, PN and CN represent the trigonometric, the polynomial and the coefficient nodes respectively. They are the nodes in a network of pointers.

exponents of the variables. Each exponent is taken to be between -128 and 128; they are packed in groups of four exponents in an integer variable (32 bits). The null exponents do not need to be stored. Together with a list of exponents the polynomial node contains a set of pointers to related nodes in the bidimensional grid making the Poisson series.

The structure of trigonometric nodes is similar to the polynomial ones, with an additional bit to signify that the function is either a sine or a cosine. The coefficient nodes are formed by the corresponding pointers and the coefficient itself. This coefficient may be a real number (type double of C) or a rational number (a pair of integers of 32 bits). The user may decide the kind of arithmetic in which to operate.

The polynomial nodes are chained in ascending lexicographic order. To be precise, we say that the node $x_0^{i_0}, \ldots, x_{n-1}^{i_{n-1}}$ precedes the node $x_0^{j_0}, \ldots, x_{n-1}^{j_{n-1}}$ when $i_0 < j_0$, or $i_0 = j_0$ and $i_1 < j_1, \ldots$

Trigonometric nodes are chained likewise with the additional provision that cosine terms precede sine terms.

Insertion of a node in a bidimensional grid is the most frequent operation for PSPC. Attentive to performance in this regard, we have looked very studiously for a good searching algorithm. The best ones, in our experience, are binary in nature; unfortunately, they cannot be applied to lists because lists are not indexed arrays. In PSPC, we made a compromise between the ordinary search in lists and the binary search. It proved to be more efficient that the pure ordinary search.

Among the problems with which we were confronted was the exchange of information with other computer programs. We solved the most pressing ones by creating functions (routines) for input and output in various formats. In particular, PSPC makes it possible to convert Poisson series to LaTeX format. We have added capabilities by which, given a Poisson series S, PSPC automatically turns the internal structure of S into a program in

either C or FORTRAN to evaluate S for any values of its polynomial and trigonometric arguments.

3. Interfacing with *Mathematica*

MathLink can provide seamless connections between *Mathematica*, your own programs and any *MathLink*-capable application (Excel, SpyGlass, MATLAB, and others). We are only interested in the option of *MathLink* which takes functions defined in an external C-program and installs them into *Mathematica*.

To see the characteristics of this procedure, let us analyse a simple example provided in Wolfram (1992). This is a procedure by which we can add two integers by means of an external program. In order to do it we must write the source code of the function `addtwo`:

```
int addtwo(int i, int j)
{
            /* Code */
 }
```

into a file called `addtwo.c` with a main program whose specifications are described in Wolfram (1992).

Furthermore, we write a template file `addtwo.tm` that identifies each external function like `addtwo` with its corresponding *Mathematica* symbol `AddTwo` and gives the pattern and the arguments of this expression to be called by *Mathematica*. Arguments of the type Integer, Real, List of Integers, List of Reals, Strings and Symbols can be passed to the external program, and only one argument of these types can be received. You can neither use pointers nor user-defined types.

The files `addtwo.c` and `addtwo.tm` must be compiled together with a special compiler of C provided with *MathLink* and then the executable program `addtwo` is created.

To use it inside *Mathematica* it is sufficient to install it by means of the expression `Install["addtwo"]`. This operation starts the external program `addtwo`. With the symbol defined in the template: `AddTwo[3,5]` the integers `3` and `5` are transferred to the external program, and it adds the numbers and transmits the result to *Mathematica* with 8 appearing in the front end of *Mathematica*.

4. PSPCLink

The goal of PSPCLink is to use from *Mathematica*, via *MathLink*, the library of C-functions PSPC.

The manipulation of series in PSPC is based on the definition of user-defined types of data. In particular, the new type of data `SERIES` does not contain the series itself, but the address of the first nodes in which the series is stored and the number of nodes. With this information we can access completely every term of the series. We used the type `SERIES` to write a routine for the addition of two series `a`, `b` to obtain a new series `c`.

```
int add_series(SERIES *a, SERIES *b, SERIES *c)
{
            /* Code   */
 }
```

The function `add_series` uses three pointers to `SERIES` as arguments and an integer as return value. This returned value is an internal control of the errors in PSPC.

The specifications about the type of arguments of *MathLink* do not permit us to use this function directly. To solve the problem we write a new function

```
int AddSeries(int n, int m, int s)
{
      return(add_series(address(n), address(m), address(s)));      }
```

in which the three series are represented by means of three integers and the address of the corresponding variables `SERIES` is given by the external array of pointers to `SERIES` called `address`.

With the same style we can write functions to perform all the operations of PSPC. Compiling the C-code of these intermediary functions together with the PSPC-code and the template file we create an external program named PSPCLink.

Taking again the addition, after installing PSPCLink into *Mathematica* we can input the expression

```
AddSeries[n,m,s]
```

This function sends the integers `n`, `m` and `s` to PSPCLink. Then, PSPCLink adds the series pointed by `address(n)` and `address(m)`, and stores the result in a place pointed by `address(s)`. *Mathematica* receives an integer that informs it about the success of the operation, but it does not receive the added series. However, this is not good *Mathematica* style; it is procedural programming, and it is not friendly to the user. For this reason we wrote a new file called `PSPCLink.m` that is a *Mathematica* package to simplify the use of PSPCLink.

Now, two kinds of Poisson series may be handled with *Mathematica*: the literal expression, and the integer number associated with a series in PSPCLink. To identify more clearly the last one we added to it the head `PoissonSeries`. In order to transfer series we built the *Mathematica* objects: `ToPoissonSeries[_]` and `FromPoissonSeries[_]`.

`ToPoissonSeries[PS]` writes the literal expression `PS` into a temporary ASCII file. PSPCLink reads this and stores the series. Eventually, `ToPoissonSeries` will return the expression `PoissonSeries[n]` with the index of the series, `n`, assigned by the system.

`FromPoissonSeries[PoissonSeries[n]]` sends to PSPCLink a command to write into a temporary ASCII file the series of index `n`. This file is read and the literal expression of the series is returned. Let us see an example

In[1]:=

```
s1 = ToPoissonSeries[(a + b) Sin[2 x - 3 y]]
```

Out[1]=

```
PoissonSeries[1]
```

In[2]:=

```
FromPoissonSeries[s1]
```

Out[2]=

```
(a + b) Sin[2 x - 3 y]
```

The use of the operator + instead of calling the procedure `AddSeries` is a better *Mathematica* style. The `TagSetDelayed` built-in function of *Mathematica* gives us the possibility of overloading the operator +.

```
PoissonSeries/:
    PoissonSeries[n_Integer] + PoissonSeries[m_Integer] :
        /*                Code to create a new object PoissonSeries[s]
           and to call AddSeries[n,m,s] to obtain its value            */
```

The rest of PSPC operations has been included in PSPCLink in the same way

In[3]:=

```
s2 = ToPoissonSeries[(a - b) Cos[2 x - 3 y]]
```

Out[3]=

```
PoissonSeries[2]
```

In[4]:=

```
s3 = s1 D[s2, x] - s2^2
```

Out[4]=

```
PoissonSeries[7]
```

In[5]:=

```
FromPoissonSeries[s3]
```

Out[5]=

```
    2          2     2            2
-3 a          b     a          3 b
----- + a b + -- + (-- + a b - ----) Cos[4 x - 6 y]
  2           2     2           2
```

Figure 2 shows the order of operations in the computation of `s3`. The number represents the index of the intermediary Poisson series generated in that process. These intermediary series are stored by PSPCLink, but usually the user does not need them. To clean the memory of useless information we created a garbage collector.

The garbage collector needs to distinguish between the *anonymous* Poisson series, intermediary series $3, 4, 5, 6$ in the example, and the *active* series, useful in later operations. In fact, not one but two different heads have been defined to this purpose. When a new series is created, *Mathematica* assigns to this series a new index `n`, creates the object

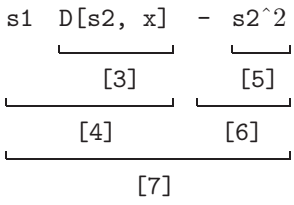

Figure 2. Order of operations.

`$PoissonSeries[n]` and adds `n` to a list of anonymous series. Only when we assign a series to a symbol, the series is converted into active series with the same index and a new head: `PoissonSeries[n]`. Together with this assignation every anonymous series is automatically deleted. This procedure forces the user to assign to a symbol every series that needs to be saved; it limits the use of the *Mathematica* symbol `%`, but allows control on the use of memory in PSPC, deleting the unnecessary intermediary results.

To change the effects of the assignation in *Mathematica* we use again the `TagSetDelayed` function

```
$PoissonSeries /:
     Set[x_, $PoissonSeries[n_Integer]] :=
          (  /* Code to delete the non-active series */
              Set[x,PoissonSeries[n]])
```

5. Efficiency of PSPCLink

To show the efficiency of PSPCLink we present here two examples. The first one is the expansion of the series

$$\left[1+\left(\sum_{i=1}^{5} x_i\right)\cos\left(\sum_{j=1}^{n} y_j\right)\right]^4.$$

We made this expansion for different numbers of trigonometric variables in two different ways: with the built-in function `Expand[_,Trig->True]` of *Mathematica* and with the PSPC function called by *Mathematica* through PSPCLink. As was expected, the resultant series is the same in both cases. The times of execution are shown in Figure 3. We can see that only in the case of no trigonometric variables the built-in function of *Mathematica* is faster. When the number of trigonometric variables increases *Mathematica* becomes slower, but PSPCLink appears to work regardless of the number of trigonometric variables; the reason is due to the packing system of variables.

The second example is a classical problem of celestial mechanics: the inversion of the Kepler equation

$$E - e\sin E = \ell.$$

In order to obtain the eccentric anomaly E as a Fourier series of the mean anomaly ℓ whose coefficients are polynomials in the eccentricity e we apply an algorithm for inversion of series due to Deprit (1979). There are simpler and more efficient algorithms to solve this example, but we use it since the method proposed by Deprit is based on the Lie

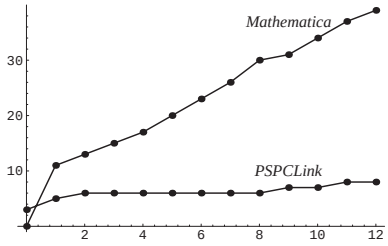

Figure 3. Time in seconds versus number n of angular variables.

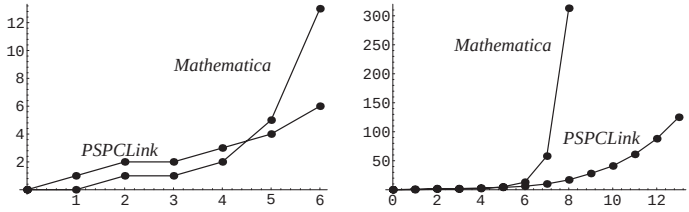

Figure 4. Time in seconds versus order of inversion.

transformation that is frequently used in the construction of analytical theories in many problems of celestial mechanics.

We compare the times of inversion of the equation for different orders using the built-in functions of *Mathematica* and PSPCLink and we show the results in Figure 4.

These figures show that *Mathematica* is faster only until fourth order. With a Macintosh Centris 660 AV with 12 MB of RAM dedicated to the *Mathematica* kernel, *Mathematica* has problems of overflow of memory after order eight; however, PSPCLink computes up to order 13 without overloading the memory.

6. Conclusions

PSPCLink increases the characteristics of *Mathematica* by using the power of PSPC in handling a Poisson series. Furthermore, it extends the possibilities of PSPC combining a Poisson series with the wide range of mathematical functions handled by *Mathematica*. The front end and the style of *Mathematica* makes the use of PSPC more friendly.

The communication tool *MathLink* made it possible to write PSPCLink. This communication tool is a good way to combine and to increase the qualities of very specialized tools and integrate them in a more general context.

PSPCLink works on any computer with *Mathematica 2.2*, *MathLink* and a C compiler compatible with *MathLink* (Wolfram, 1992). There is no special requirement in hardware, in particular the dynamic allocation of PSPC permits the use of all free RAM memory. Obviously, the maximum size of the series handled by PSPC depends on the amount of RAM memory of the computer.

Both PSPC and PSPCLink can be obtained by sending an e-mail to the first author.

Acknowledgements

We are very grateful to Dr Deprit for his valuable suggestions. This work has been supported in part by the Ministerio de Educación y Ciencia (DGICYT Projects # PB95-0807 and # PB95-0795). The authors are cited in alphabetical order.

References

Abad, A., San-Juan, J.F. (1993). PSPC: A Poisson Series Processor coded in C. *Dynamics and Astrometry of Natural and Artificial Celestial Bodies, Poznam, Poland*, pp. 383–389.

Broucke, R.A. (1989). A Fortran-based Poisson series processor and its applications in celestial mechanics. *Celestial Mech. Dynam. Astron.* **45**, 255–265.

Brumberg, V.A., Chapront J. (1973). Construction of a general planetary theory of the first order. *Celestial Mech. Dynam. Astron.* **8**, 335–356.

Dasenbrock, R.R. (1982). A FORTRAN-based program for computerized algebraic manipulation. *NRL Report 8611.*

Deprit, A. (1979). Note on Lagrange's inversion formula. *Celestial Mech. Dynam. Astron.* **20**, 325–327.

Deprit, A. (1981). Celestial mechanics: never say no to a computer. *J. Guidance Control* **4**, N. 6, 577–581.

Deprit, A. (1990). Processing poisson series in parallel,*J. Symbolic Comput.* **10**, 179–201.

Deprit, A., Henrard, J., Rom, A. (1970). Analytical lunar ephemeris: Delaunay's theory. *Astron. J.* **76**, 269–272.

Henrard, J. (1989). A survey of Poisson series processors. *Celestial Mech. Dynam. Astron.* **45**, 245–253.

Osácar, C., Palacián, J. (1994). Decomposition of functions for elliptic orbits. *Celestial Mech. Dynam. Astron.* **60**, 207–223.

Palacián, J. (1993). *Teoría del satélite artificial: Armónicos teserales y su relegación mediante simplificaciones algebraicas.* Publicaciones Seminario Matemático García Galdeano, Serie II, Universidad de Zaragoza. Spain.

Ricklefs, R.L., Jefferys, W.H., Broucke, R.A. (1983). A general precompiler for algebraic manipulation. *Celestial Mech. Dynam. Astron.* **29**, 179–190.

San-Juan, J.F. (1996). *Manipulación Algebraica de Series de Poisson Aplicación a la Teoría del Satélite Artificial.* Publicaciones de la Universidad de Zaragoza.

Stroustrup, B. (1987). What is object-oriented programming. *Lecture Notes in Comput. Sci.* **276**, 51–70.

Wolfram, S. (1988). *Mathematica. A System for Doing Mathematics by Computer.* Addison Wesley.

Wolfram, S. (1992). *MathLink. Reference Guide.* Technical Report. Wolfram Research.

Originally received 31 July 1995
Accepted 3 February 1997